Naturalness of Attention: Revisiting Attention in Code Language Models

Mootez Saad and Tushar Sharma
{mootez,tushar}@dal.ca
Dalhousie University
Halifax, Nova Scotia, Canada

ABSTRACT

Language models for code such as CodeBERT offer the capability to learn advanced source code representation, but their opacity poses barriers to understanding of captured properties. Recent attention analysis studies provide initial interpretability insights by focusing solely on attention weights rather than considering the wider context modeling of Transformers. This study aims to shed some light on the previously ignored factors of the attention mechanism beyond the attention weights. We conduct an initial empirical study analyzing both attention distributions and transformed representations in CodeBERT. Across two programming languages, Java and Python, we find that the scaled transformation norms of the input better capture syntactic structure compared to attention weights alone. Our analysis reveals characterization of how CodeBERT embeds syntactic code properties. The findings demonstrate the importance of incorporating factors beyond just attention weights for rigorously understanding neural code models. This lays the groundwork for developing more interpretable models and effective uses of attention mechanisms in program analysis.

KEYWORDS

Attention Analysis, Language Models of Code, Norm Analysis, Interpretability

ACM Reference Format:

Mootez Saad and Tushar Sharma. 2023. Naturalness of Attention: Revisiting Attention in Code Language Models. In *Proceedings of 46th International Conference on Software Engineering (ICSE 2024).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Obtaining effective representations of source code is crucial for many program analysis tasks such as code search, code completion, and program translation. Earlier code representation approaches, such as Code2Vec [2] and Code2Seq [1] demonstrated initial progress in learning distributed vector representations of code. However, these methods are limited in their modeling capacity and do not fully capture the rich semantics of code. To overcome the limitations in early code representation techniques, Transformer-based [16] neural models have emerged as a promising paradigm for learning effective code representations. Inspired by their success in natural language processing, architectures such as BERT [4] have been adapted to code representation, leading to the emergence of Language Models of Code (LMC), such as CodeBERT [5], GraphCodeBERT [7], CodeT5 [19], and Code Llama [13]. The self-attention mechanism powering Transformers provides stronger representational capabilities compared to earlier architectures such as Recurrent Neural Networks (RNNs). This has enabled Transformer-based models to establish new state-of-the-art results across a variety of software engineering tasks that involve source code analysis, processing, and manipulation by learning better semantic representations of programs.

Despite such advances, a major limitation of LMCs is their black box nature and lack of interpretability. While models such as CodeBERT show impressive performance on downstream tasks, it is unclear which properties of code they capture or learn internally.

To address these interpretability issues, an emerging area of research involves analyzing and probing these complex neural networks through attention visualization and representation analysis. Sharma *et al.* [15] found BERT models trained on code exhibit key attention differences from natural language - namely, higher focus on identifiers over special tokens like `[CLS]` and more localized context. Wan *et al.* [18] investigated the encoded syntactic patterns within the attention distributions of CodeBERT and GraphCodeBERT. However, in the realm of natural language, Kobayashi *et al.* [10] note that attention weights alone may not reveal the full perspective of the patterns learned by the model. Based on the *naturalness* property of software [8], techniques effective for analyzing natural language models may also lend insight into source code models. This motivates revisiting attention-based analysis by incorporating the factors that were previously ignored when analyzing LMCs.

In this paper, we revisit the mathematical formulation of the Multihead Attention (MHA) [10] to illustrate how it is composed of two factors: *attention weights* and the *transformation of input*. Given this new reformulation, we perform a trend analysis of the Transformer layers of CodeBERT on Java and Python to study the differences between these two factors. We show how including the previously ignored effect leads to a better alignment with the syntactic properties of source code compared to the attention weights.

We make the following contributions:

- First, we perform a layer-wise analysis to study the trends of attention weights **and** the transformation of input for an LMC (*i.e.,* CodeBERT). To the best of our knowledge, this is the first study that emphasizes the need to consider the transformation of input along with attention weights in the context of an LMC.
- Second, we compare the capacity of the two factors to capture the syntactic properties of source code.

We make the replication package, including code and data, available online [14].

ICSE 2024, April 2024, Lisbon, Portugal
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND AND MOTIVATION

The core component of the Transformer architecture is the Multiheaded Attention (MHA), which is composed of multiple Self-Attention (SA) heads. For instance, in CodeBERT, which is based on RoBERTa's [11] architecture, each MHA layer is composed of 12 SA heads. Let h denote the number of heads, and X be the input sequence of length n, where each token is embedded in $\mathbb{R}^{d'}$ ($d' = 64$ in CodeBERT). In each head, a sequence is projected into three matrices: *Query* ($Q_{n\times d'}$), *Key* ($K_{n\times d'}$) and *Value* ($V_{n\times d'}$). Formally, these matrices are defined as follows:
For $i \in [1 \dots h]$

$$Q^{(i)} = X \cdot W_Q^{(i)},\ K^{(i)} = X \cdot W_K^{(i)},\ V^{(i)} = X \cdot W_V^{(i)} \tag{1}$$

The attention matrix A is computed by applying the `softmax(.)` function on the result of the multiplication of the *Query* and *Key* matrices[1], scaled by the square root of their dimension d'.

$$\mathrm{A}^{(i)} = softmax(\frac{Q^{(i)} {K^{(i)}}^T}{\sqrt{d'}}) \tag{2}$$

Then, the attention matrix is multiplied by the *Value* matrix to obtain the attention output z,

$$z^{(i)} = \mathrm{A}^{(i)} \cdot V^{(i)} \tag{3}$$

Finally, concatenating the output of each head and multiplying it by a weight matrix $W^O_{hd'\times hd'}$ gives the output of the MHA layer.

$$Z_{\text{MHA}} = [z_1; \dots ; z_h]_{n\times hd'}, \tag{4}$$

$$Y_{\text{MHA}} = Z \cdot W^O \tag{5}$$

Y_{MHA} can be reformulated given the linearity of matrix multiplication. To build the intuition, let us consider the calculation of the entry located at the 1$^{\text{st}}$-row and 1$^{\text{st}}$-column of Y_{MHA}. It is done by taking the dot product of the 1$^{\text{st}}$-row of Z_{MHA} and the 1$^{\text{st}}$-column of W^O.

$$Y_{\text{MHA}}[1,1] = \sum_{i=1}^{hd'} Z_{\text{MHA}}[1,i] W_O[i,1] \tag{6}$$

We can decompose Equation (6) into h summations,

$$\begin{aligned} Y_{\text{MHA}}[1,1] = & \textstyle\sum_{i=1}^{d'} Z_{\text{MHA}}[1,i] W_O[i,1] \\ & + \dots \\ & + \textstyle\sum_{i=hd'-d'+1}^{hd'} Z_{\text{MHA}}[1,i] W_O[i,1] \end{aligned} \tag{6}$$

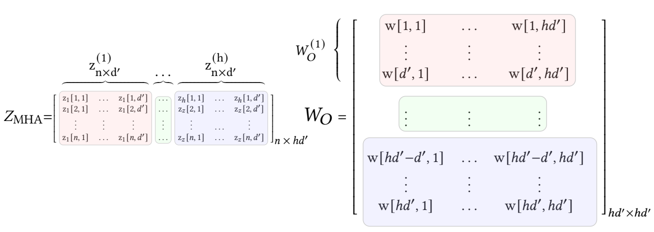


Figure 1: Multiplication of the concatenated outputs of each SA head with the weight matrix. The highlighted blocks are the components that are attributed to each SA head.

[1] In Vaswani *et al.* [16], this multiplication includes an optional mask to mask out certain tokens such as padding tokens. We omit this for the sake of simplicity.

By extension and with reference to Figure 1, we can express Y_{MHA} as the sum of h matrices calculated from the multiplication of the submatrices from Z_{MHA} and W_O. This entails that Equation (5) can be rewritten as follow,

$$Y_{\text{MHA}} = Z_{\text{MHA}} \cdot W^O = \sum_{i=1}^{h} z^{(i)} \cdot W_O^{(i)} \tag{9}$$

and if we plug in Equation (3) and Equation (1) we obtain,

$$\begin{aligned} Y_{\text{MHA}} &= \sum_{i=1}^{H} \mathrm{A}^{(i)} \cdot V^{(i)} \cdot W_O^{(i)} \\ &= \sum_{i=1}^{H} \mathrm{A}^{(i)} X \cdot W_V^{(i)} \cdot W_O^{(i)} \\ &= \sum_{i=1}^{H} \mathrm{A}^{(i)} f^{(i)}(X) \end{aligned} \tag{10}$$

We can see from this reformulation that the information Y_{MHA} holds is the result of the contribution from *two* factors: the attention weights A *and* the transformation $f(.)$ applied on the input X. A token $t_i \in X$ can have a high attention weight $\alpha_i > 0$ and, at the same time, a low contribution from its transformation $||f(t_i)|| \approx 0$ [10]. *This implies that the properties deduced from the analysis of MHA cannot solely be attributed to the attention weights*. The previous studies that performed attention analysis of Language Models of Code (LMC) concentrated on probing the attention weights to see the type of patterns they exhibit and how well they align with the properties of source code [15, 18].

Motivated by this argument, we would like to revisit the attention analysis of LMC by considering the missing factor, *i.e.,* the scaled transformation $||\alpha f(x)||$.

The primary objective is to thoroughly comprehend the attention mechanism's properties when applied to source code. To this end, we conduct a preliminary empirical study and answer the following research questions:

RQ1: How do the general trends across layers between attention weights α and the scaled transformations norms $\|\alpha f(x)\|$ compare?

RQ2: How does $\|\alpha f(x)\|$ align with the syntactic structure of source code compared to attention weights?

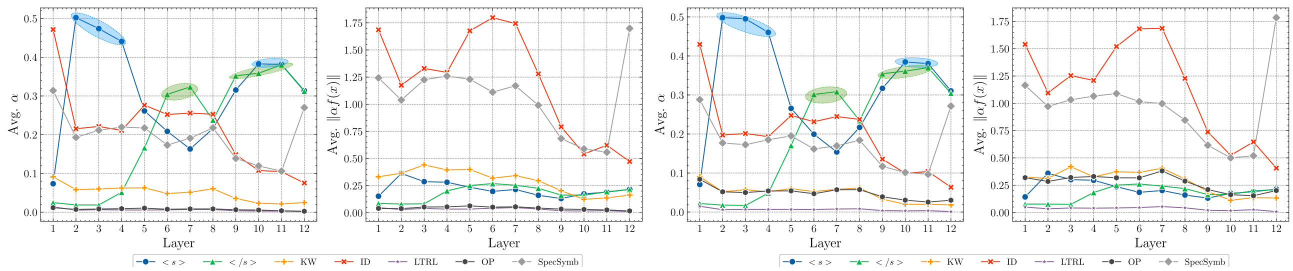
3 EXPERIMENTS AND RESULTS

In this section, we will present our methodology and the conducted experiments to answer the research questions stated above.

3.1 Methodology

Models: In this study, we considered CodeBERT [5], a pretrained language model of code that adopts the architecture and pertaining strategy as RoBERTa [11]. We chose such a model to follow Wan *et al.* [18]. In addition, it is one of the earliest LMCs which has spawned many follow-up works. Analyzing it provides a strong baseline for future comparative studies on other related models.

It consists of 12 Transformer [16] layers, each encompassing 12 self-attention heads. It was trained on a set of bimodal instances (*i.e.,* pairs of natural language and programming languages), across six programming languages from the CodeSearchNet dataset [9].



(a) Python **(b) Java**

Figure 2: Variation of the average α and $\|\alpha f(x)\|$ values of each token category across layers in CodeBERT. Abbreviations: Keywords (`KW`), Identifiers (`ID`), Literals (`LTRL`), Operators (`OP`), Special Symbols (`SpecSymb`). Note that `<s>` and `</s>` are the classification and separation tokens respectively. In other BERT variants, they are represented by `[CLS]` and `[SEP]`.

Data: We used CodeSearchNet dataset [9] to create corpora for two programming languages: Java and Python. Each corpus consists of 5,000 randomly sampled code snippets with lengths less than 512 tokens.

3.2 RQ1—Trend Analysis

Through this research question, we aim to investigate the trends, at a macro level, in the behaviours of the attention weights α versus the scaled norms $\|\alpha f(x)\|$ across the layers of CodeBERT.

Approach: From each self-attention head, we extracted the attention weights α, transformation norm $||f(x)||$ and scaled transformation norm $\|\alpha f(x)\|$ matrices, for each instance. Since CodeBERT was trained using a *WordPiece* tokenizer [20], each word can be tokenized further into subtokens. Given that our analysis was carried out at a word level, we follow the same procedure done by Clark *et al.* [3] and convert token-token maps to word-word maps by taking the average of a word's subtokens. To make our analysis granular, we group tokens by their categories: *Keywords*, *Identifiers*, *Literals*, *Operators* and *Special Symbols*. Each category is defined according to each programming language grammar specification (Java [12] and Python [6]).

Results: Figure 2 depicts the variation of the average attention weights and the average scaled transformation norms across each layer for each token category in both datasets. Aligned with the findings of Kobayashi *et al.* [10] and Clark *et al.* [3], and in contrast to the results of Sharma *et al.* [15], the special tokens displayed higher average attention compared to other token types. Specifically, `<s>` had the highest average attention between Layers 2 and 4, which then decreased until Layer 7. Then, its attributed attention kept on increasing until Layer 10. Interestingly, the drop in `<s>`'s attention between Layers 5 and 7 appeared to transfer to `</s>`, whose average attention peaked within this range. This pattern held across programming languages, with similar trends in both Java and Python corpora, though minor differences arose in precise attention values. For instance, `<s>` attention remained constant between Layers 2 and 3 for Java, whereas it slightly declined in the Python dataset.

However, this pattern is different when calculating the *scaled transformation norms* $\|\alpha f(x)\|$. The contribution of these tokens is lower compared to other categories such as *Identifiers* and *Special Symbols*. This indicates, that similar to BERT, when CodeBERT does not find information in the input, it assigns higher attention values to these tokens given that the attention weights should sum up to 1 (due to the `softmax(.)` function).

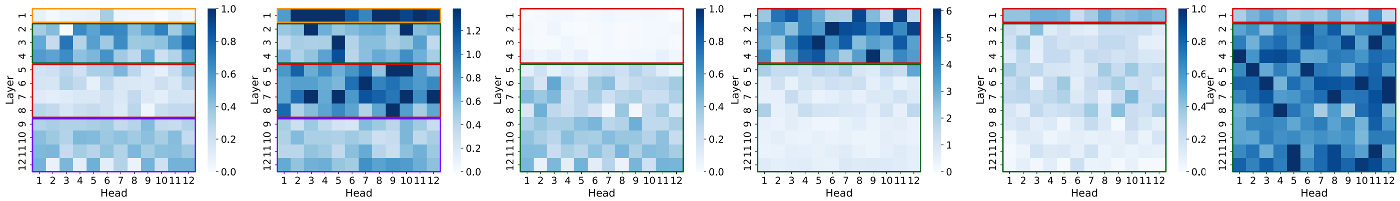
Although the *ranking* of each token category at each layer appears to be consistent between α and $\|\alpha f(x)\|$, there is some contrast between the patterns observed at each layer. For example, if we look at the trend of attention weights of the *Keyword* tokens in the Python dataset in Figure 2a, we see that the attention values drop between Layer 1 and Layer 2 and remain relatively constant between Layers 2 and 4. In contrast, the values of $\|\alpha f(x)\|$ between Layers 1 and 3 are increasing for this category. This contrast effect is also observed for other types such as *Identifiers*. Generally, in both datasets, we see that in some layers, when the attention weights are constant (*e.g.,* L2-L4 and L5-L8) the scaled transformation norms exhibit either a peak or a decline. One explanation can be the *cancelling* effect of α and $||f(x)||$ that was mentioned in Section 2 , hence, the contrast between α and $\|\alpha f(x)\|$. Figure 3 further illustrates this cancelling effect for the special tokens `<s>` and `</s>`, and the *Literals* category.

> **Summary**: The results show that the behavior of attention weights and the scaled transformation norms $\|\alpha f(x)\|$ differ significantly. The two components of Multiheaded Attention *i.e.,* α and $||f(x)||$, often exhibit a cancellation effect. Such contrast entails that including other variables, *i.e.,* the transformation norm $||f(x)||$, when performing attention analysis, might lead us to more comprehensive and explainable results. In other words, extending the analysis to regions other than attention weights might reveal additional insights about the language model's capacity to model the relations and properties of source code.

3.3 RQ2—Syntactic Alignment

In this section, we analyze the syntactic properties that are embedded in $\|\alpha f(x)\|$ compared to those in the attention weights.

Approach: Vig et al. [17] proposed a metric, $p_\alpha(g)$, that measures the agreement between an attention map (*i.e.,* the attention matrix or weights) and a property map generated by an indicator function g. The function $g(i, j)$ returns 1 if a given property exists between two tokens i and j, 0 otherwise. Wang *et al.* [18] defined g to return 1 if the pair (i, j) share the same parent in the Abstract Syntax Tree



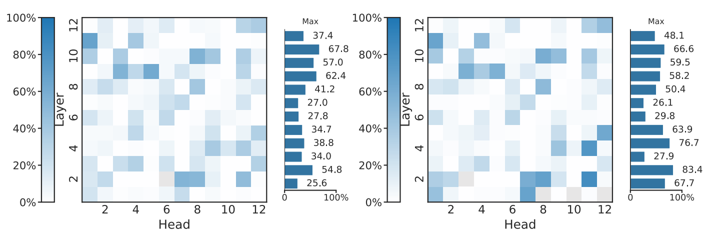
(a) <s> **(b) </s>** **(c) LTRL**

Figure 3: Attention (α) and transformation norm maps ($||f(x)||$) for <s>, </s>, and *Literals* from the Java dataset. For each type of token, the left and right figures refer to the attention and norm map respectively. Regions that show the contrast relation are highlighted with the same colour.

(AST) of a code snippet x. Their intuition was that attention defines the closeness of each pair of code tokens. This score is formally defined in Equation (11),

$$p_\alpha(g) = \frac{\sum_{\mathbf{x}\in\mathbf{X}} \sum_{i=1}^{|\mathbf{x}|} \sum_{j=1}^{|\mathbf{x}|} f(i,j) \cdot \mathbb{1}_{\alpha_{i,j}>\theta}}{\sum_{\mathbf{x}\in\mathbf{X}} \sum_{i=1}^{|\mathbf{x}|} \sum_{j=1}^{|\mathbf{x}|} \mathbb{1}_{\alpha_{i,j}>\theta}} \tag{11}$$

where $\mathbb{1}_{\alpha_{i,j}>\theta}$ is an indicator function that selects high-confidence attention weights ($\theta = 0.3$ in [17, 18]). In other words, $\mathbb{1}_{\alpha_{i,j}>\theta}$ evaluates to 1 if $\alpha_{i,j} > \theta$, and 0 otherwise. Equation (11) sums over all token pairs (i, j) where the attention $\alpha_{i,j} > \theta$. It counts how many of these high-confidence attention pairs connect tokens that are syntactically related according to the AST (*i.e.,* $f(i,j) = 1$) over the dataset. Dividing this count by the total number of high-confidence pairs gives the proportion of attention connections that align with the AST structure. This proportion indicates how well the attention matches syntactic relationships.



(a) $p_\alpha(g)$ **(b)** $p_{\alpha f(x)}(g)$

Figure 4: Agreement between the AST structure and the attention and scaled norm matrices on the Python dataset for all heads. We scale $\|\alpha f(x)\|$ maps within the $[0\ldots1]$ range. The bar plots at the right of each subfigure show the maximum agreement at each layer.

Results: Figure 4 illustrates the degree each head in each of CodeBERT's layers aligns with the code's syntactic properties by considering attention weights and $\|\alpha f(x)\|$. Overall, it appears that the scaled transformation embeds better the syntactic properties of source code than the bare attention weights. Interestingly, it even captures such properties starting at earlier layers. For instance, we see that in Layer 2, the maximum agreement percentage is 83.4% compared to 54.8% in the attention maps, and it is ×2.6 more aligned in the first layer. The same trend is also observed in the majority of the remaining layers, whether in the middle (*e.g.,* Layer 4, 76.7% vs 38.8% and Layer 5, 63.9% vs 34.7%) or at the end (Layer 10, 59.5% vs 57% and Layer 12, 48.1% vs 37.4%). However, there are layers where the attention weights exhibit a higher alignment than $\|\alpha f(x)\|$. Notably, in Layers 3, 7 and 10 (34% vs 27.9%, 27% vs 26.1%, and 67.8% vs 66.6%). However, the alignment with attention weights is significantly lower than the alignment with scaled transformation

Summary: Generally, the scaled transformation norms show a higher alignment with the syntactic properties of source code. However, there are regions where the attention weights model better such properties. These observations contribute to our understanding of how the attention mechanism embeds implicit code patterns.

4 CONCLUSIONS AND FUTURE WORK

In this work, we have revisited the mathematical definition of MHA from prior works in natural language. We showed how the attention mechanism is not merely composed of the attention weights. The preliminary findings indicate that incorporating scaled transformation norms provides new perspectives on what code properties are captured by attention.

The presented work can be extended in a variety of directions. The first extension point is to investigate how these findings could vary across other programming languages and models. Applying the same methodology to models such as GraphCodeBERT [7] and CodeT5 [19] and other languages such as JavaScript and Go will test the generalizability of our results. Along similar lines, we aim to evaluate models trained with techniques that are more programming language-oriented. CodeBERT, despite being trained on NL-PL pairs, was pre-trained with the same objective as RoBERTa to model natural language. On the other hand, GraphCodeBERT encodes data flow paths in its input, which captures more source code properties other than its sequential nature. Similarly, CodeT5 uses an identifier-aware pre-training task that allows the model to determine which code tokens are identifiers and to recover them when they are masked. Determining if specialized training objectives modify attention behaviors will clarify how different representations are learned. The goal is to connect training procedures with resultant attention patterns.

REFERENCES

[1] Uri Alon, Omer Levy, and Eran Yahav. 2019. code2seq: Generating Sequences from Structured Representations of Code. In *International Conference on Learning Representations*.

[2] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. Code2vec: Learning Distributed Representations of Code. *Proc. ACM Program. Lang.* 3, POPL, Article 40 (jan 2019), 29 pages. https://doi.org/10.1145/3290353

[3] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What Does BERT Look at? An Analysis of BERT's Attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, Florence, Italy, 276–286. https://doi.org/10.18653/v1/W19-4828

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[5] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[6] Python Software Foundation. Accessed on 2023-08-22. *2. Lexical analysis*. https://docs.python.org/3/reference/lexical_analysis.html

[7] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie LIU, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCode{BERT}: Pre-training Code Representations with Data Flow. In *International Conference on Learning Representations*.

[8] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the Naturalness of Software. In *Proceedings of the 34th International Conference on Software Engineering* (Zurich, Switzerland) *(ICSE '12)*. IEEE Press, 837–847.

[9] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *CoRR* abs/1909.09436 (2019). arXiv:1909.09436 http://arxiv.org/abs/1909.09436

[10] Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui. 2020. Attention is Not Only a Weight: Analyzing Transformers with Vector Norms. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 7057–7075. https://doi.org/10.18653/v1/2020.emnlp-main.574

[11] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[12] Oracle. Accessed on 2023-08-22. *Chapter 2. Grammars*. https://docs.oracle.com/javase/specs/jls/se7/html/jls-2.html#jls-2.3

[13] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]

[14] Mootez Saad and Tushar Sharma. 2023. SMART-Dal/norm-analysis-clm: v1.1.0. https://github.com/SMART-Dal/norm-analysis-clm

[15] Rishab Sharma, Fuxiang Chen, Fatemeh Fard, and David Lo. 2022. An Exploratory Study on Code Attention in BERT *(ICPC '22)*. Association for Computing Machinery, New York, NY, USA, 437–448. https://doi.org/10.1145/3524610.3527921

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[17] Jesse Vig, Ali Madani, Lav R. Varshney, Caiming Xiong, richard socher, and Nazneen Rajani. 2021. {BERT}ology Meets Biology: Interpreting Attention in Protein Language Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=YWtLZvLmud7

[18] Yao Wan, Wei Zhao, Hongyu Zhang, Yulei Sui, Guandong Xu, and Hai Jin. 2022. What Do They Capture? A Structural Analysis of Pre-Trained Language Models for Source Code. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 2377–2388. https://doi.org/10.1145/3510003.3510050

[19] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *EMNLP*.

[20] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). arXiv:1609.08144 http://arxiv.org/abs/1609.08144